DELIVERABLE 2

KALASHNIKOVA POLINA 0293706

INTRODUZIONE

- Ci è stato chiesto di realizzare un software in java che combinando jira e git hub con l'uso di machine learning predice se una classe sarà buggy oppure no nella futura release.
- Per accedere a Jira sono utilizzate le API di Jira, per github e weka – plugin di Java.
- IDE di sviluppo Visual Studio Code
- Progetti da analizzare: Bookkeeper e Tajo

VERSIONI

- Per prima cosa usando API di JIRA ed il codice visto a lezione recuperiamo i dati delle versioni e issues da Jira.
- Per le versioni prendiamo tutti i record di Jira relativi al progetto, creiamo una lista numerata con tutte le versioni, estrapolandoli da Json ricevuto da Jira.

Index	Version ID	Version Name	Date
1	12324252	0.2.0	20.11.2013
2	12324253	0.8.0	01.05.2014
3	12325340	0.9.0	21.10.2014
4	12328654	0.10.0	09.03.2015
5	12329684	0.10.1	29.06.2015
6	12333355	0.11.1	03.02.2016
7	12334786	0.11.2	07.04.2016
8	12335664	0.11.3	18.05.2016
9	12329002	0.11.0	27.10.2016
10	12332471	0.12.0	11.09.2019

JIRA ISSUES

- Con API di JIRA e query: issuetype=bug AND (status=resolved OR status=closed) AND resolution=fixed prendiamo tutti issues bug fixed da Jira.
- Per ogni issue trovato estrapoliamo :
- Campo "created" (data di creazione)- serve per trovare OV (opened version) usando la lista di versioni.
- Array „fixVersions" - serve per trovare FV (fixed version) prendendo il massimo.
- Campo „resolutiondate" - se fixed versions non e' presente, usiamo questa data per trovare la fixed version.
- Array „versions" (affected versions)- serve per trovare AV usando la lista di versioni create in precedenza.
- Settiamo IV (injected version) come prima (minima) tra AV.
- Injected e Affected version possono essere vuoti.
- AV non puo' includere FV.
- OV e FV devono essere presenti.

JIRA ISSUES: INJECTED VERSION

- Poiche' IV puo' essere vuota, usiamo la tecnica proportion per predirla
- Usiamo incremental per determinare valore di proportion basato su IV, OV e FV presenti.
- sum += (double) (fv - iv) / (fv - ov + 1)
- Proportion = Sum / Numero di tickets usati
- Predicted IV = FV - proportion * (FV - OV + 1)
- Dopo aver applicato proportion il numero di classi Buggy e' aumentato per entrambi progetti:
- 15,35% per Bookkeeper e 8,31% per Tajo

JIRA ISSUES: AFFECTED VERSIONS

- Ora ricalcoliamo affected versions: come valore tra injected e fixed version, fixed escluso
- Se injected version e fixed sono uguali, affected version puo' essere vuoto e successivamente questi tickets verranno eliminati da analisi
- Ora abbiamo tabella con issues con le relativi versioni

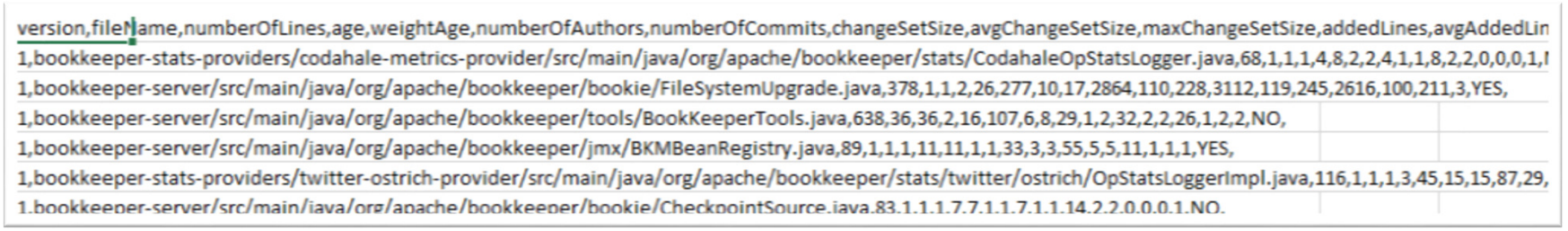
Index	Issue	IVersion	OVersion	AVersion	FixVersion
41	TAJO-1110	0.9.0	0.9.0		0.9.0
42	TAJO-1111	0.8.0	0.9.0	0.8.0	0.9.0
43	TAJO-1113	0.8.0	0.9.0	0.8.0	0.9.0
44	TAJO-1119	0.8.0	0.9.0	0.8.0, 0.9.0	0.10.0
45	TAJO-1126	0.9.0	0.10.0	0.9.0	0.10.0
46	TAJO-1139	0.9.0	0.10.0	0.9.0	0.10.0
47	TAJO-1146	0.8.0	0.10.0	0.8.0, 0.9.0, 0.10.0, 0.10.1, 0.11.1, 0.11.2, 0.11.3	0.11.0
48	TAJO-1147	0.10.0	0.10.0	0.10.0, 0.10.1, 0.11.1, 0.11.2, 0.11.3	0.11.0
49	TAJO-1150	0.9.0	0.10.0	0.9.0	0.10.0
50	TAJO-1154	0.9.0	0.10.0	0.9.0	0.10.0

GITHUB: COMMITS

- Scorrendo la lista con le versioni, prendiamo il nome di ogni versione
- Con plugin di git per java prendiamo tutti i commit per il progetto usando **tag di github** con il nome della versione
- La lista di commit ottenuta incrociamo con la lista di issues, cercando il **numero di issue nel titolo** di commit
- Facciamo parsing del commit trovato per avere dei lavori che ci serviranno per analisi successiva e creiamo la lista con i commits.
- Valori di interesse: **data del commit, authore, versione**
- Ogni commit inoltre ha una lista dei file comittati. Scorriamo la lista dei files elaborandola per il nostro scopo e creiamo una lista nuova con dei files che hanno valori che ci interessano (numero righe modificate, aggiunte, tipo del commit add/change/delete)
- Alla fine abbiamo 2 liste concatenate: commits con i relativi files comittati.

CALCOLO DELLE METRICHE

- Preparazione del dataset: raggruppo la lista dei files per il nome(path), avro' per ogni file una lista collegata che uso per i calcoli: count per il calcolo delle medie, raggruppo per authors – numero di autori, sommo le righe modificate, ecc.
- Metriche calcolate:
- Eta' (trovando la data del commit iniziale), eta' pesata, numero di righe modificate (media e massimo), righe aggiunte (media e massimo), churn lines (added – deleted), numero autori, numero comits, ecc.
- Setto buggy sul true se la versione del file appartiene ad affected versions.
- Creo un file csv e un arff con i valori calcolati per ogni classe e ogni versione:



```
version,fileName,numberOfLines,age,weightAge,numberOfAuthors,numberOfCommits,changeSetSize,avgChangeSetSize,maxChangeSetSize,addedLines,avgAddedLin
1,bookkeeper-stats-providers/codahale-metrics-provider/src/main/java/org/apache/bookkeeper/stats/CodahaleOpStatsLogger.java,68,1,1,1,4,8,2,2,4,1,1,8,2,2,0,0,0,1,
1,bookkeeper-server/src/main/java/org/apache/bookkeeper/bookie/FileSystemUpgrade.java,378,1,1,2,26,277,10,17,2864,110,228,3112,119,245,2616,100,211,3,YES,
1,bookkeeper-server/src/main/java/org/apache/bookkeeper/tools/BookKeeperTools.java,638,36,36,2,16,107,6,8,29,1,2,32,2,2,26,1,2,2,NO,
1,bookkeeper-server/src/main/java/org/apache/bookkeeper/jmx/BKMBeanRegistry.java,89,1,1,1,11,11,1,1,33,3,3,55,5,5,11,1,1,1,YES,
1,bookkeeper-stats-providers/twitter-ostrich-provider/src/main/java/org/apache/bookkeeper/stats/twitter/ostrich/OpStatsLoggerImpl.java,116,1,1,1,3,45,15,15,87,29,
1,bookkeeper-server/src/main/java/org/apache/bookkeeper/bookie/CheckpointSource.java,83,1,1,1,7,7,1,1,7,1,1,14,2,2,0,0,0,1,NO,
```

WEKA

- Separo il dataset di prima per ogni versione (un csv e arff per ogni versione) che sono 10 versioni per Tajo e 14 versioni per Bookkeeper per facilitare walk forward
- Prendo solo la prima meta delle versioni
- Applico i classificatori: **naive base**, **random forest** e **ibk** ai dataset usando la tecnica **walk forward**
- Per ciascun classificatore si applica: la feature selection **best first** o **no selection**; la tecnica di sampling **over sampling**, **under sampling**, **smote** o **no sampling**; il cost sensitive classifier di tipo **sensitive threshold**, **sensitive learning** o **nessuna**
- La combinazione produce 72 walk forward in totale (3*2*4*3)
- Per ogni combinazione abbiamo N record corrispondente al numero delle versioni usate (4 = 5 -1 per Tajo e 6 = 7-1 per Bookkeeper)

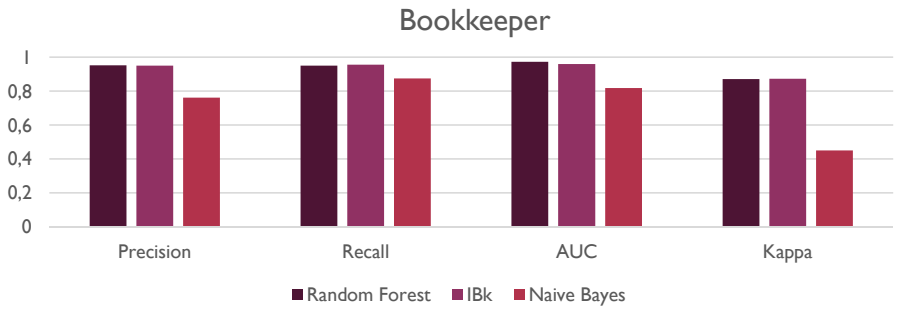
WEKA – WALK FORWARD

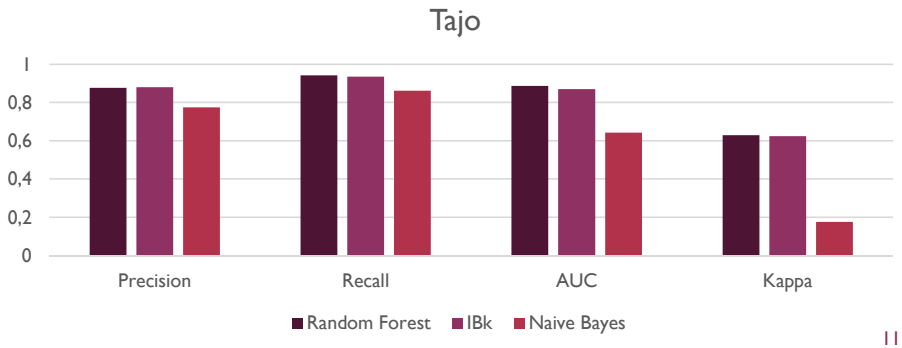
Per ogni versione x si prende come training set la versione (x e (x-1)) e per la testing set la versione x+1

- Esempio, Tajo: in totale 5 versioni:
- Run1: train versione 1 – test versione 2
- Run 2: train versione 1, 2 – test versione 3
- Run 3: train versione 1, 2, 3 – test versione 4
- Run 4: train versione 1, 2, 3, 4 – test versione 5
- Si applicano i classificatori
- Si fa la media di ogni walk (cioe' la media di 4 run se prendiamo Tajo)

ANALISI CLASSIFICATORI: CONFRONTO ACCURATEZZA

- Uso le metriche di **Precision, Recall, AUC** e **Kappa** di ogni classificatore prendendo i valori con **No Sempling, No Selection, No cost sensitive**.



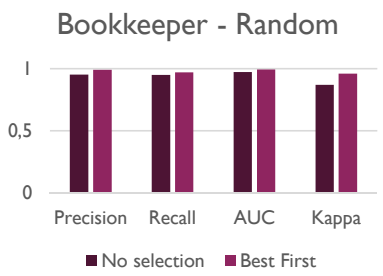


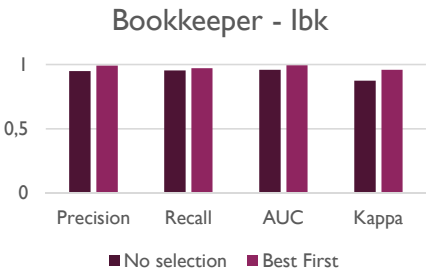
ANALISI CLASSIFICATORI: CONFRONTO ACCURATEZZA

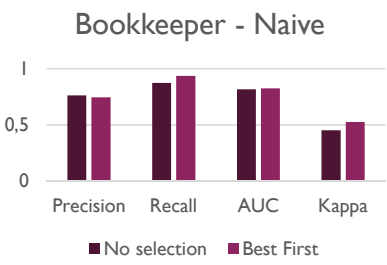
- La differenza visibile c'e' solo su **Kappa** del progetto **Tajo** del classificatore **Naive Bayes**
- Kappa sempre positive
- Tutti i valori per le metriche di accuratezza sono abbastanza alti, vuol dire che anche senza applicare le tecniche particolare I nostri classificatori riescono a predire con una buona percentuale di esattezza.
- Questo vuol dire che il nostro dataset strutturato bene, e le metriche che ci sono siano davvero importanti per predire se la classe sia buggy o no

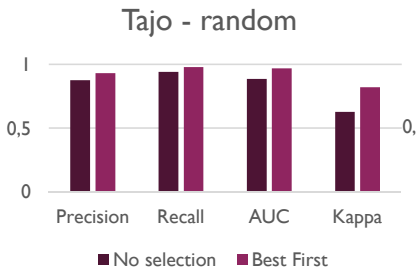
ANALISI CLASSIFICATORI: FEATURE SELECTION

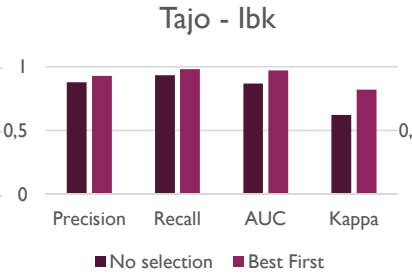
- No Cost, No sampling
- Best First e' sempre migliore
- Naive Bayes e' sempre peggiore

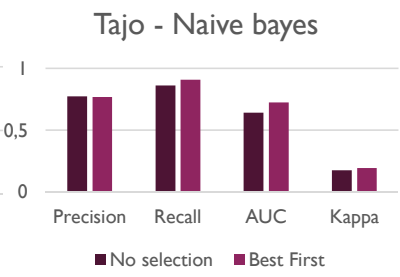










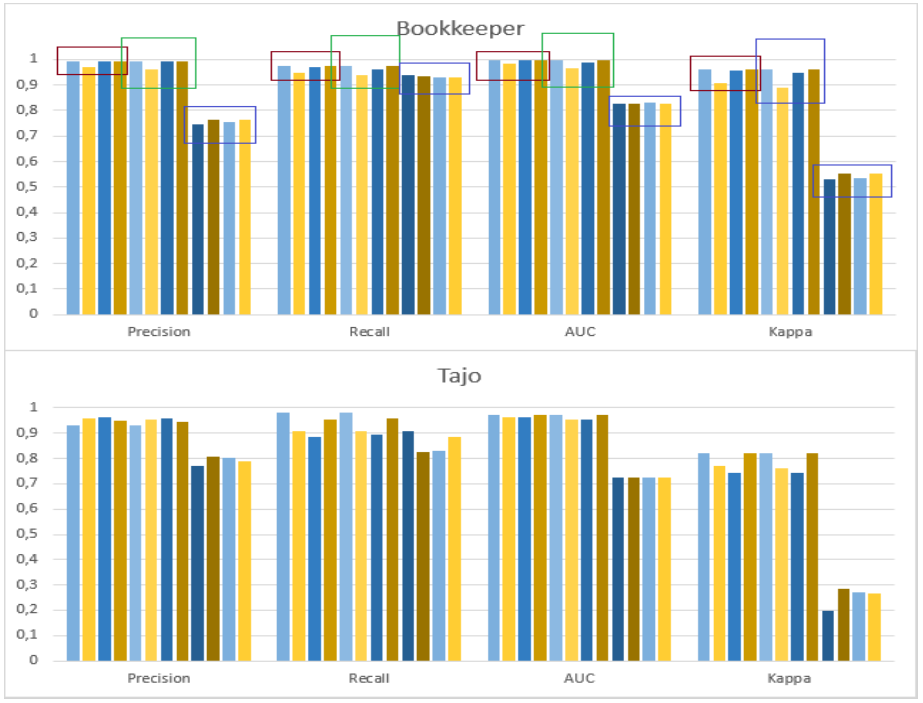


ANALISI CLASSIFICATORI: FEATURE SELECTION

- Best first risulta sempre migliore, useremo questo nelle successive valutazioni.
- Per Tajo I 3 migliori features sono: Numero dei autori, eta', N of bugs
- Mi aspettavo anche altri come size e numero di line modificate, se tolgo N of bugs, allora Weka prende anche Changed lines, I valori non cambiano molto, se tolgo Changed lines Weka prende N of commits e valori non cambiano, ma se tolgo numero di autori, I valori scendono notevolmente. Vuol dire che questi valori sono molto rappresentativi almeno nel mio dataset.
- Per Bookkeeper sono: N commits, changed set size e N of bugs

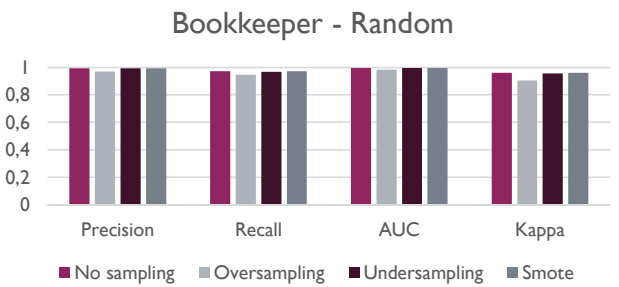
ANALISI CLASSIFICATORI: SAMPLING

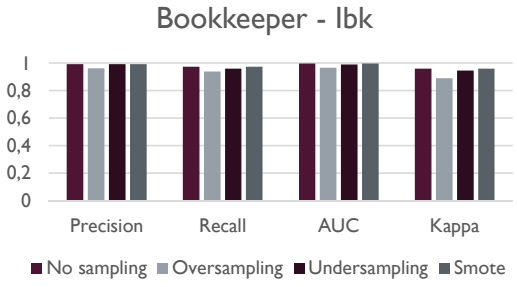
- Facciamo grafico per tutti i classificatori: **Random Forest, Ibk, Naive Bayes** con **Best First** e **No Cost**
- **Best First** perche' nel confronto precedente era sempre migliore di «no selection».
- Da questo grafico si vede che i valori delle ultime 4 colonne (sono No sampling, Oversampling, Undersampling, Smote) per ogni tripla dei classificatori sono sempre minori di altri, questo significa che Naive Bayes e' sempre peggiore per entrambi progetti. Lo eliminiamo dall'analisi successiva.

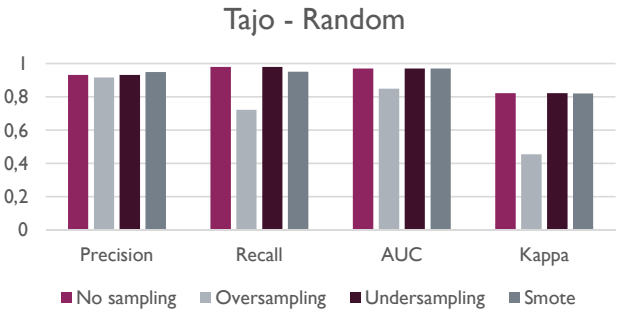


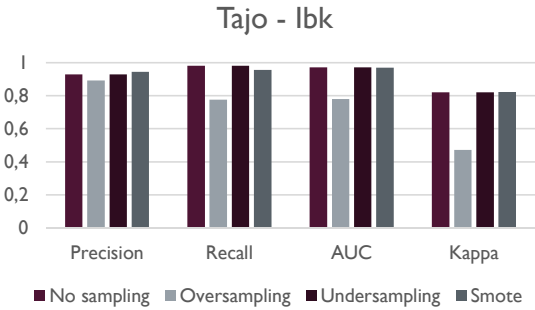
ANALISI CLASSIFICATORI: SAMPLING

- Best First con Random Forest e Ibk,
- Naive Byes abbiamo escluso
- No Cost







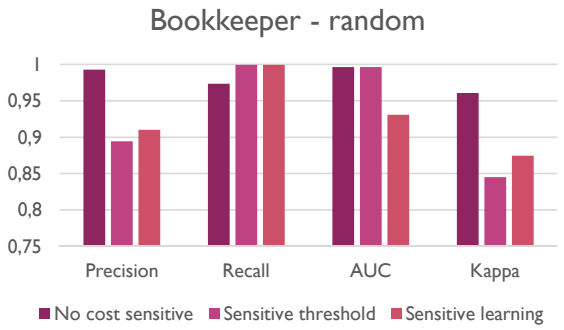


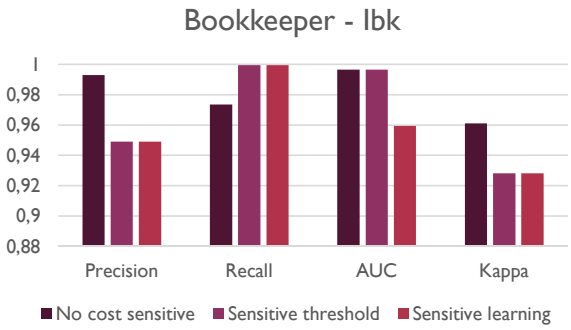
ANALISI CLASSIFICATORI: SAMPLING

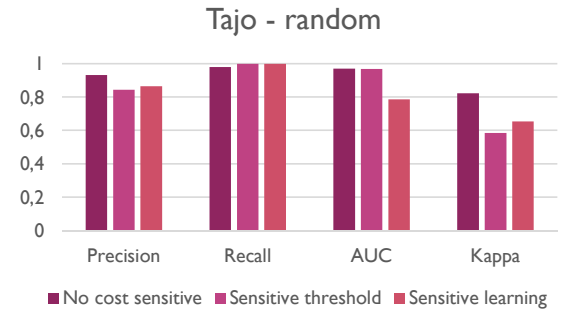
- Per Bookkeeper Oversampling risulta sempre peggiore
- Altri 3 non cambiano molto
- Anche per il Tajo sembra la stessa tendenza.

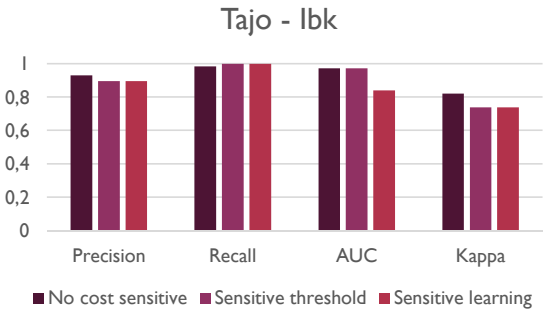
ANALISI CLASSIFICATORI: COST SENSITIVE

- Best First con Random Forest e Ibk,
- Naive Byes abbiamo escluso
- No cost sensitive, Sensitive threshold, Sensitive learning









ANALISI CLASSIFICATORI: COST SENSITIVE

- Per precision e Kappa cost sensitive ha peggiorato, meglio senza nessun cost sensitive
- Sensitive learning ha migliorato Kappa, ma ha peggiorato AUC.
- Sensitive threshold ha migliorato recall e non ha avuto nessuna influenza sul precision
- Il vincitore e': nessuno. Per precision, kappa e AUC meglio senza nessun cost sensitive
- Solo per recall meglio Sensitive threshold.

CONCLUSIONE

- Naive Bayes e' sempre peggiore, ma era da aspettarsi, visto che da' ottimi risultati sul analisi del testo, ma nel nostro caso era valore si/no o numerico.
- Bookkeeper e' sempre meglio di Tajo.
- La migliore combinazione risulta Best First come tecnica di selection, No sampling o Undersampling per il sampling. Anche se Smot non e' molto lontano da Undersampling.
- La tecnica di cost sensitive ha migliorato solo valore di Recall.
- Il classificatore Random Forest risulta leggermente migliore del Ibk, probabilmente perche' l'albero delle decisioni come algoritmo corrisponde meglio al nostro scopo, invece del vicino piu' vicino (KNN in Weka si chiama Ibk)

LINKS

- GitHub:
- https://github.com/404NotFoundPK/ISW2-DELIVERABLE2
- Dataset e output weka:
- https://github.com/404NotFoundPK/ISW2-DELIVERABLE2/tree/main/output
- Sonar:
- https://sonarcloud.io/dashboard?id=404NotFoundPK_ISW2-DELIVERABLE2